Rollback - **return to some safe state, restart process from that state**

3-2>

When deadlock is detected, the system must recover either

by termination some of the deadlocked process (데드락 걸린 프로세스 종료)

or by preempting resources from some of the deadlocked processes (데드락 걸린프로세스의 자원 선점해버림)

<Deadlock Detection>

i) Allow system to enter deadlock state

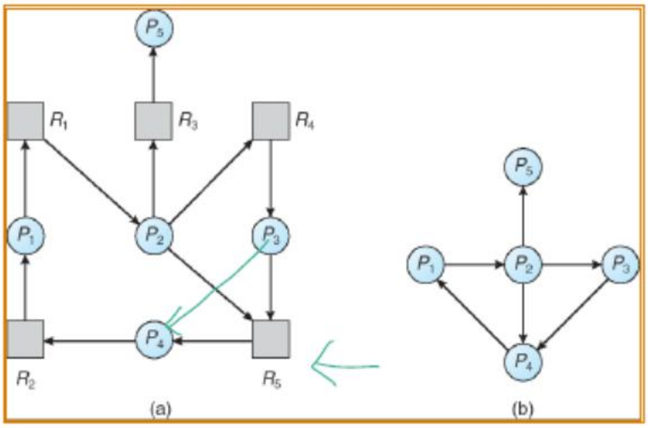
ii) Detection algorithm

iii) Recovery scheme

<Single Instance of each Resource Type>

- How to detect deadlock using resource-allocation graph
- Maintain wait-for graph from resource-allocation graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j
- Periodically invoke an algorithm that searches for a cycle in the graph
  - If a cycle exist → deadlock
- An algorithm to detect a cycle in a graph requires an order of n^2 operations,
  - where n is the number of vertices in the graph

<Resource-Allocation Graph and Wait-for Graph>



Resource-Allocation Graph Corresponding Wait-for graph

- How to detect deadlock in a multiple instances of a resource type?
- Available: A vector of length m indicates the number of available resources of each type.
- Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process.
- Request: An n x m matrix indicates the current request of each process.
  - If Request [i,j] = k, then process P_i is requesting k more instances of resource type R_j.

Detection Algorithm (Cont.) – modified safety algorithm

1. Let Work and Finish be vectors of length m and n, respectively.
   Initialize: Work = Available
   For i=0, 1, .. , n-1, if $Allocation_i \neq 0$, then Finish|[i] = false;
   Otherwise, Finish [i] = true.
2. Find an index i such that both:
   (a) Finish [i] == false
   (b) $Request_i \leq Work$
   If no such i exists, go to step 4.
3. $Work = Work + Allocation_i$
   Finish[i] = true
   go to step 2.
4. If Finish [i] == false, for some i, $0 \leq i \leq n-1$, then the system is in deadlock state.
   Moreover, if Finish[i] == false, then P_i is deadlocked.

Algorithm requires an order of O(m x n^2) operations to detect whether the system is in deadlocked state. ♪

ㅠㅠ ㅠㅠ

- Five processes P_0 through P_4;
- three resource types
  - A (7 instances), B (2 instances), and C (6 instances).
- Snapshot at time T_0:

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in Finish[i] = true for all i.

- P_2 requests an additional instance of type C.

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

걍빠ㄴ

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes' requests.
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

<Detection-Algorithm Usage>

- When, and how often, to invoke depends on"

  - How often a deadlock is likely to occur?

  - How many processes will need to be rolled back? : one for each disjoint cycle

- If detection algorithm is invoked arbitrarily(독단적으로),

  - there may be many cycles in the resource graph

  - and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

<Recovery from Deadlock> : several alternatives

1. operator에게 데드락 일어났다고 알려서 manually 처리하게 하기

2. Process Termination : circular wait 깨게 하기 위해 종료시킴

 i) Abort all deadlocked processes.

 ii) Abort one process at a time until the deadlocked cycle is eliminated

3. Resource Preemption : deadlocked process으로부터 자원 선점(preempt) 하기

 three issues in resource preemption

 1) Selecting a victim - minimize cost

 2) **Rollback** - **return to some safe state, restart process from that state**

 3) Starvation - same process may always be picked as victim.